The Comparison of Audio Using Signal Processing and Various Machine Learning Models
Part 1 - A CNN Approach

Kaii Bijlani

With Dr Joe Xiao

Abstract

Top companies are now using machine learning models more than ever to target consumers with appropriate product recommendations. In music apps such as Spotify, recommendations are made based on factors such as music genre, artist, or listener demographics. Learning models such as k-nearest neighbors or approximate nearest neighbors are used to compare songs in their vector space with the factors listed. The goal of this project is to explore how direct audio can be fed into machine learning models and made into discrete output that can be compared with other songs. The project is focused around the method of signal processing, which essentially converts a continuous sound wave into discrete time-based signals, or samples. This can be achieved through the use of Spectrograms, which is a 2D representation of an audio wave. The Spectrogram can then be fed into a multilayer network to analyze its contents. We focus on utilizing a classic CNN model architecture to process the values of the Spectrogram into numerical values, which can then directly be compared with other songs. A combined CNN-LTSM model is also discussed as a means of more accurately processing Spectrogram values.

1. Introduction:

Companies such as Spotify use standard recommendation techniques to generate lists of recommended songs to their users. They tend to utilize decision trees or other classification techniques to generate vector values of a song based on data given by other users and the song itself. There are various types of data that song recommendation systems may analyze [1]. One popular data type commonly analyzed in recommendation systems is user listening habits. Listening habits include how often the user skips throughout a song, how far they skip along, average viewing time, and other points of reference for the user's interaction with one song. In recommendation systems, machine learning can be utilized to draw conclusions of how the user interacts with a certain song to recommend similar songs. The method of comparing songs to other songs in order to know which songs to recommend to a user is where it gets more complex.

This paper focuses on analyzing the content of a song/audio file as a means of comparison. The method of signal processing is proposed to convert a continuous audio signal into discrete time-based samples. To do so, spectrograms are used to create a visual representation of an audio signal over a time-space. From there, data can be converted to discrete values via matrix manipulation of an image, which then can be used to generate predictions about the spectrogram and thus predictions about the original audio/song. This can be achieved through a Convolutional Neural Network architecture, which will parse through different layers of the image and determine patterns, even if they may be in different locations. Finally, the

output vector will be converted to a probability vector and classified via one-hot encoded label matching. Loss and accuracy will then be analyzed.

2. Datasets and Modules

To demonstrate a working CNN framework, we propose the standard CIFAR-10 dataset [2]. This will provide a vast array of images and truth labels to analyze a classification CNN model. The CIFAR-10 dataset provides 60,000 images to work with, of which 10,000 are declared as testing images and the rest as training images. Thus, there is no need to declare a testing and training proportion as a parameter. Each of the 60,000 images is classified as one of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. The dataset provides 6,000 images for each label, rendering bias obsolete for evaluating test images. Each image is size 3x32x32 (3072 flattened) meaning each image has a length and width of 32 pixels, and each pixel has a value for the 3 RGB color channels.

For audio classification, we propose the GTZAN dataset [3]. This dataset provides 10,000 30 second long songs, each under one of 10 classification labels: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. The dataset makes it easy to work with small music samples of equal size to explore the relationship between the content of a song and how it is classified.

From PyTorch we use Torchaudio [4] as the module utilized for audio processing. The tools provided in this module make it simple to visualize signal processing and allow easy access to hyperparameters and other values, such as the sampling rate. PyTorch is used for creating the CNN model. Other datasets include Pandas, Matplotlib, NumPy, and Librosa. We also use Scikit-Learn as a means of evaluating model accuracy via confusion matrices and accuracy scores.

3. Signal Processing, Spectrograms, and Convolutional Neural Networks

To start, it is necessary to understand the process of audio processing before considering the usage of machine learning and regression techniques to analyze spectrograms.

3.1. Signal Processing

Sound waves are continuous, which are impossible to feed into a deep learning system unless made discrete. Signal processing [5] helps convert continuous time signals to discrete time-based samples. A discrete sample $s(t)$ is the value of a signal at time t. Figure 1 shows a continuous signal curve being split up into discrete signals. The amount of samples to take is determined by the sampling rate, dependent on its time between samples T. In Figure 1, T would be one second

as there is a one second gap between the samples taken. This divides the signal into T bins, each bin having an area value of $\int_{t_1}^{t_2} S(t)dt$. The sampling rate is determined by the following formula:

Equation 1: $f_s = \frac{1}{T}$

The units are in samples per second, or Hertz. 44.1 kHz is most commonly used as the sampling rate, as most music applications use 44100 Hertz, though this project uses both 44100 Hertz and 22050 Hertz.

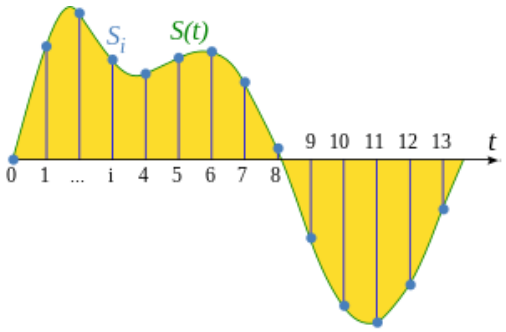

Figure 1: A continuous signal *S(t)* being split into discrete samples S_i. The number of samples to take per second is equal to the sampling rate f_s.

3.2. Nyquist Sampling Theorem

The Nyquist Sampling Theorem [6] states that for a continuous signal *S(t)* with a frequency no higher than *B* hertz, then the sampling rate should be larger than *2B* hertz. In other words, the sampling rate should be higher than 2 times the maximum frequency. This fundamental theorem in signal processing primarily applies to band-limited signals, or signals that go to 0 hertz beyond some time *t*. If there are any more restrictions on the signal curve, then the Nyquist Sampling Theorem doesn't hold.

3.4 Waveforms

The waveform [7] of a signal is the shape of the signal curve as it varies with time. Waveforms are commonly used in signal processing to visualize audio signals, but are rarely ever used as input to any audio classification models. This is because the waveform does not show the energy of bins, or the "loudness" of the bin in a time interval. Torchaudio can create waveforms using the load() function. This function accepts the audio file name as a parameter (e.g. 'testfile.wav') and returns a tuple with the sampling rate value and the waveform, in the form of a Tensor. Figure 2 shows the waveform of the song "Her Majesty" by The Beatles, the .wav file being just below 28 seconds. As can be seen, the time-axis ends at about 28 seconds, and the oscillation of the signal curve throughout the song represents the amplitude of the signal. There are also two

channels of the signal, but since they are nearly identical it is feasible to ignore the difference in amplitudes (see *Advantages of Two Channels in Waveform Generators* [8] for more information).

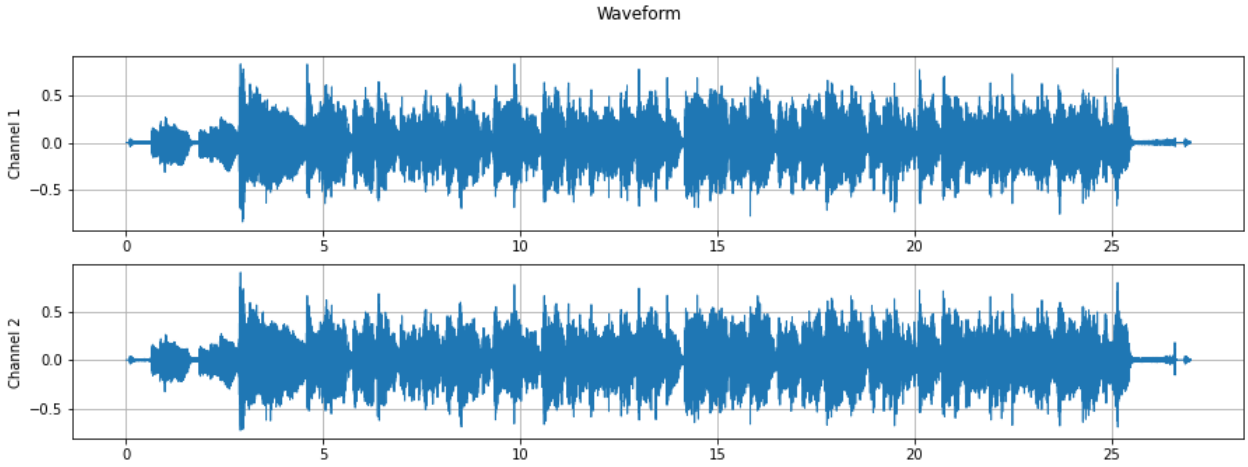

Figure 2: A two-channel waveform representation of the song "Her Majesty" by The Beatles.

3.3. Spectrograms

Spectrograms[9] are used in signal processing to convert an audio signal to a visual image. They are a heatmap representation of a waveform and are used to show the energy of signal frequencies as they vary with time. The energy of a frequency bin can be defined to be the "strength" or "loudness" of that bin. Typically with spectrograms, higher energy bins are represented by brighter colors and vice versa, making it easy to pinpoint the signal's strength as it varies with time. Torchaudio.transforms is a module that provides an easy framework for spectrograms via their spectrogram class, with multiple types of spectrograms. Figure 3 shows a decibel spectrogram of the song "Her Majesty" by The Beatles. The spectrogram was initially a power spectrogram created by torchaudio.transforms, but the Librosa library was used to convert it to a decibel spectrogram via the power_to_db() function, which transforms the values of the spectrogram based on the following equation:

$$\text{Equation 2: } dbSpec = 10 * log_{10} \frac{powerSpec}{k}$$

Where powerSpec is the power spectrogram and k is an optional scalar with a default value of 1. The purpose of the transformation is to convert the units of a power spectrogram to the units of a decibel spectrogram. The units of a power spectrogram are in amplitude(metres)^i, where i is the power of the spectrogram. For torchaudio, this default power is two, so the units would be amplitude^2. The decibel conversion converts the units to decibels, making it easy to see the energy changes in terms of the "loudness" of the signal, as audio strength is measured in decibels. In essence, decibels are a logarithmic conversion of amplitude and make the spectrogram better fitted to visualize.

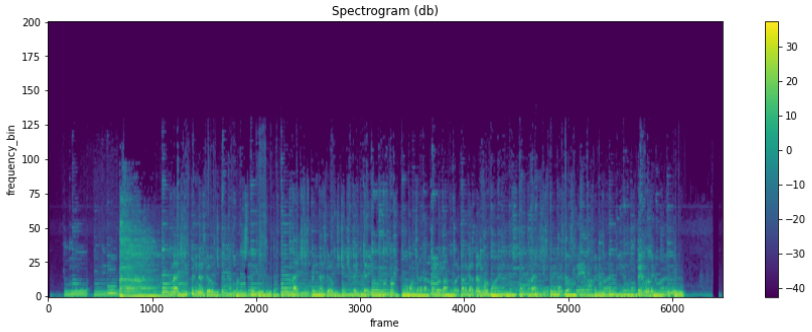

Figure 3: A db-Spectrogram of the song "Her Majesty" by The Beatles. The sampling rate is 48,100 Hertz.

3.4. Mel Spectrograms

The Mel spectrogram [10] is the result of a mel-scale transformation of a spectrogram. The mel-scale is meant for logarithmically converting frequency values to frequency values that are easier to see and visualize. The mel-scale essentially converts all the original frequency values to new values that are representative of their relative distances to other frequencies. Torchaudio creates Mel spectrograms via the MelSpectrogram() class, where the argument *n_mels* represents the number of mels to separate the frequency bins into. The number of mels makes the frequencies evenly spaced on a listening scale, making it much easier to visualize and learn from in machine learning models. For this paper, the Mel spectrogram is the input type for the CNN model.

3.5. Convolutional Neural Networks

Traditional neural networks have feed-forward, dense linear layers (meaning each layer is connected to each subsequent layer). This means that there is a fixed input and a fixed output state, and there are only linear transformations. In a neural network, there is an input layer, hidden layers, and an output layer. Each hidden layer has a number of neurons, each connected to the next layer with their corresponding weights. There is also an activation curve and a bias applied by the computer via backpropagation. The output of a hidden layer follows this equation:

Equation 3: $$output_i = bias + Activation(\sum_{i=1}^{k} weight_i * input_i)$$

k is the number of neurons, meaning the output is the activated sum of all the neurons and their weights, plus a bias. There are many types of activation curves to be used. This paper utilizes the ReLU activation function, which takes the maximum value from 0 to the input value.

The loss of the model is the error of the network. After an epoch is completed, the computer goes back and updates the weight to new values to approach a lower error, or a lower loss. The loss

function can be represented by *L(t)*, and thus to find the minimum value of the function calculus must be used to find derivatives. This process is known as gradient descent [11], which is done by backpropagation [12], or using the chain rule to find the partial derivatives of loss with respect to the weights, and thus finding new values for the weights that will minimize the loss. Figure 4 shows a model of gradient descent.

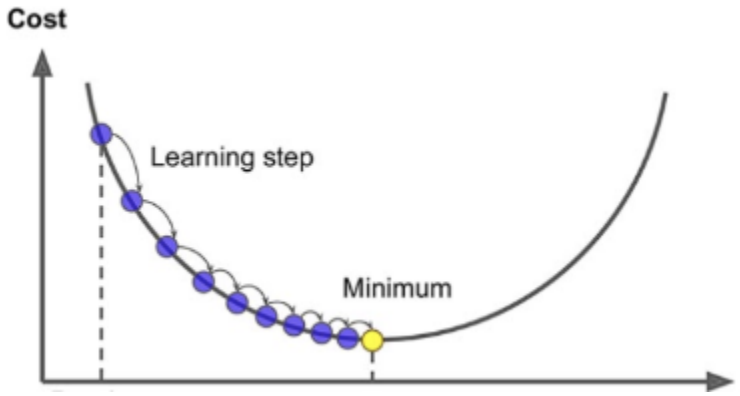

Figure 4: A model of gradient descent

The metric used to measure the loss for this paper is CrossEntropyLoss. The following equation shows the calculation of CrossEntropyLoss, where x is the input and y is the output:

Equation 4: $$l_n = -\ weight_{y_n} * log \frac{exp(x_n, y_n)}{\sum_{class=1}^{Classes} exp(x_n, class)}$$

Typically, in classification problems, a softmax activation curve is needed to convert a vector of numbers into a vector of probability. This is done by dividing the exponentiated product of an input and output by the sum of all exponiated inputs and classes, but notice that this is already done by the CrossEntropyLoss metric.

The optimizer is the function that changes the weights in a network. The optimizer used for this paper is Stochastic Gradient Descent (SGD). This optimizer makes it easy to not get stuck on gradient descent via the momentum argument, which is a moving average of past gradients to prevent oscillation between two gradient values.

A Convolutional Neural Network (CNN) [13] is used to recognize patterns amongst inputs, which is the reason most image classification problems use it. With regular neural networks, it is difficult to recognize patterns amongst an image because it is not designed to do so.

A CNN has the typical neural network linear layers with additional convolutional and pooling layers. In a CNN, a kernel extracts information in an image via convolutions. The kernel size is smaller than the image size, and the step size is the step that the kernel moves every time it is

done with calculation. The next step is to pool the significant data from that convolution. The most popular type of pooling is max pooling, which takes the maximum values of parts of a convoluted layer. Finally, the data is flattened from a Tensor to a vector in order to be fed into a linear neural network. Figure 5 shows an image going through a convolutional neural network, where the first two layers show a convolutional-pooling system.

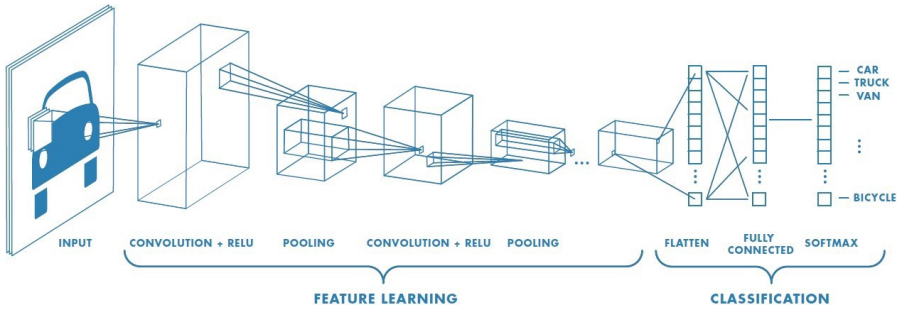

Figure 5: An image going through a convolutional neural network. The feature learning points out patterns from the image, and these are fed into the classification model.

A sample convolutional neural network was created and tested on the CIFAR-10 dataset. This network had two convolutional and pooling layers, and four linear (dense) layers. Figure 6 shows the results of the model. The model was trained by randomly selected images from the CIFAR-10 dataset over 10 epochs, and the results were generated by dividing the number of correctly predicted images by the number of images. This value turned out to be 62%, which is fairly standard for a simple convolutional network.

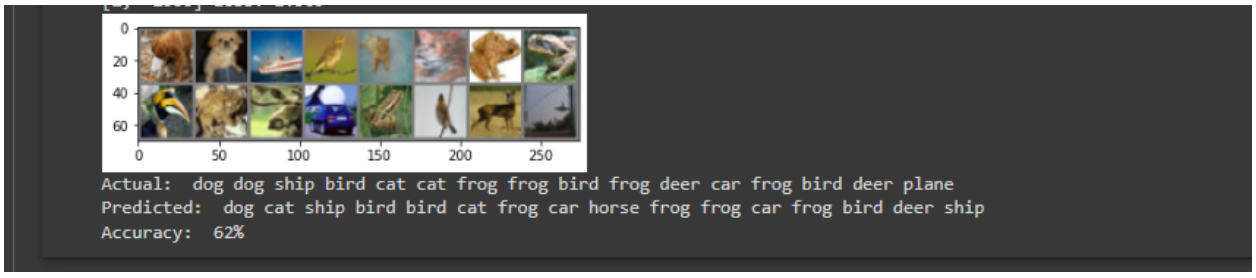

Figure 6: Results of a simple convolutional network on the CIFAR-10 dataset.

4. Results, Conclusion, and Future Implementation

4.1. Results and Conclusion

The model used for this paper first preprocessed the data by converting the .wav audio file to a Mel spectrogram, and converted to a decibel scale system. The Mel spectrogram was created at a sampling rate of 22,050 Hertz and fed into a CNN, with 5 convolutional and max pooling layers. The model also had a dropout layer, which randomly removes nodes to prevent overfitting.

Finally, the convoluted, flattened spectrogram was fed into a 5 layer linear node system and converted to a vector of values, which was then converted to probabilities via CrossEntropyLoss and back propagated against to optimize loss. For training, around 30 randomly selected GTZAN songs were fed into the network, first over 25 epochs and then over 80 epochs. The model was tested on around 50 randomly selected GTZAN songs, first after being trained with 25 epochs and then after being trained with 80 epochs. Figure 7 and 8 show the results for 25 epochs and 80 epochs, respectively.

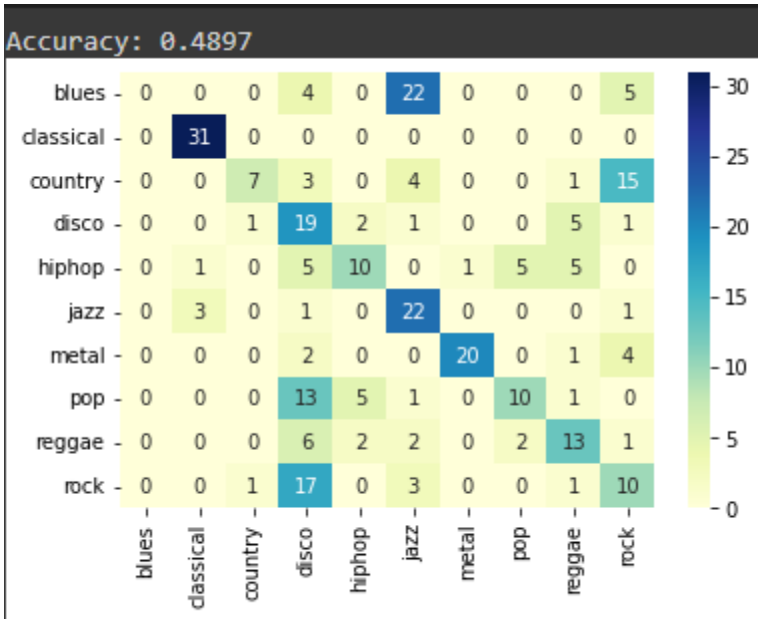

Figure 7: A confusion matrix of the predicted vs actual genres for a music classification model over 25 trained epochs. The horizontal axis represents predicted genres and the vertical axis represents actual genres. The model has an accuracy of 48.97%.

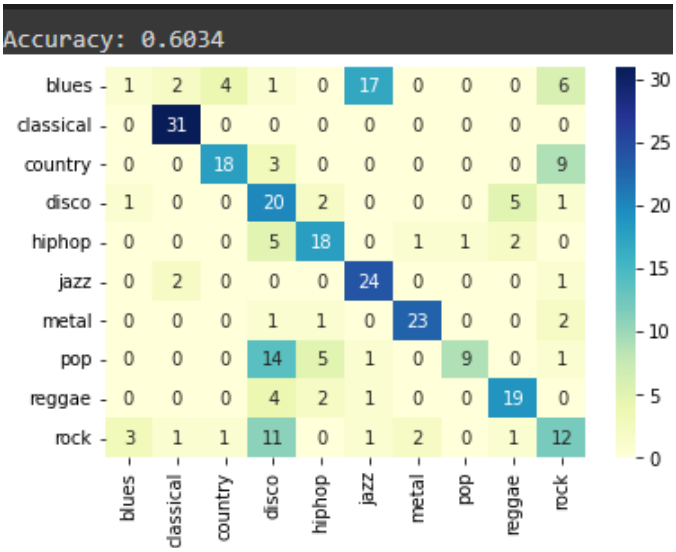

Figure 8: A confusion matrix of the predicted vs actual genres for a music classification model over 80 trained epochs. The horizontal axis represents predicted genres and the vertical axis represents actual genres. The model has an accuracy of 60.34%.

The resulting confusion matrices show the relationship between predicted genres vs actual genres of GTZAN songs. The direct relationship between the number of epochs and the accuracy indicate that the model is strong and resilient against overfitting and underfitting.

The genre with the least number of predictions was blues. This could be explained due to the nature of blues songs; typically, blues songs were a combination of multiple music types, making it hard for both a human and computer to differentiate it from other genres. This is also explained in the results by analyzing the jazz predicted column. As can be seen, 22 blues songs were predicted as jazz songs for the 25 epoch model and 17 blues songs were predicted as jazz songs for the 80 epoch model. This could be due to the similarity in musical rifts between the two genres: both jazz and blue incorporate soft, slow music with a lot of constant rhythm and melody.

Classical was the genre that was best predicted. This might be due to the fact that classical is very easy to differentiate from the other genres: it has lots of piano and less background music, as well as no lyrics. Thus, the model did not need to be trained very long for it to classify classical songs correctly.

Other genres that the model found hard to predict correctly include pop and rock, both of which were predicted as disco. This is explainable due to the fast, loud nature of disco, making it easy for a computer to classify it as rock or pop. However, it is interesting to note that the computer didn't predict pop or rock songs as disco songs. The most surprising genre mix-up was country and rock. The 25 epoch confusion matrix shows 15 country songs being predicted as rock, and 9 country songs predicted as rock for the 80 epoch confusion matrix. A possible explanation for

this could be the common factor of guitars and vocal lyrics among the two genres, though they are still very different.

4.2. Future Implementations via Recurrent Neural Networks (RNNs) or Long Short Term Memory Network (LSTMs)

Traditional CNNs may not be fully accurate in linking one part of a song with previous parts, and recognizing a pattern that may be crucial in genre classification. Recurrent Neural Networks (RNNs) [14] help with this problem. In an RNN, input is converted to a sequence of one element per cell which produces an output (hidden state), which is then combined with the next element in the next cell as the input. This continues until the end of the input sequence. RNN cells are consistent throughout the implementation, meaning they are reused, but with different inputs. Classification would only require one final output (softmax for probability vector), whereas something like natural language processing would require an output at every cell to generate new text (based on the previous words). Since the final output of a RNN cell is based on previous outputs (inputs to the cell), the RNN accounts for every single input in the sequence. The following equation is an example hidden layer for a RNN cell:

Equation 5: $hidden_t = tanh(weight_{hidden} * hidden_t - 1 + weight_{input} * input_t)$

Notice the extra weight times input part, which differentiates a RNN from a normal neural network. Backpropagation would work a bit differentially: gradients are taken at different time steps and backpropagate normally, but they add together when different time step gradients meet (i.e. new input gradients add with output gradients). RNNs would be particularly useful in a task like audio classification because they can use the signal as a discrete sequence of bins, and generate assumptions about the patterns between different bins.

Another type of model to consider is the Long Short-Term Memory model (LSTM) [15]. An LSTM is similar to a recurrent neural network, meaning it combines new inputs and previous outputs as inputs. However, LSTMs can remember long term information, whereas RNNs can't. This is extremely useful in audio classification where patterns can be found even if they are far apart in a song. RNNs attempt to link each input with each previous input, as humans do in things like sentences. The problem arises when the gap between relevant inputs in an input sequence is too large, and we have a long-term dependency, rendering LSTMs helpful. The cell state is the primary function in the LTSM, running linearly with three gateways for optional information. The gates are made of a sigmoid activation function, outputting a probability vector with values between 0 and 1. Each of the values tell how much new input information should be weighted in the cell state, 0 being none and 1 being all. Figure 9 and 10 show the difference between a standard RNN and an LSTM model.

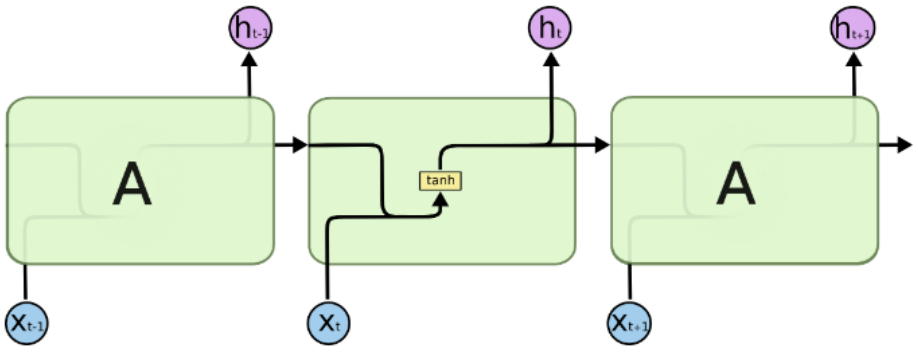

Figure 9: Model of a RNN. Each *X* is a new input, and each *h* is an output at a time. There is only one layer in a standard RNN cell.

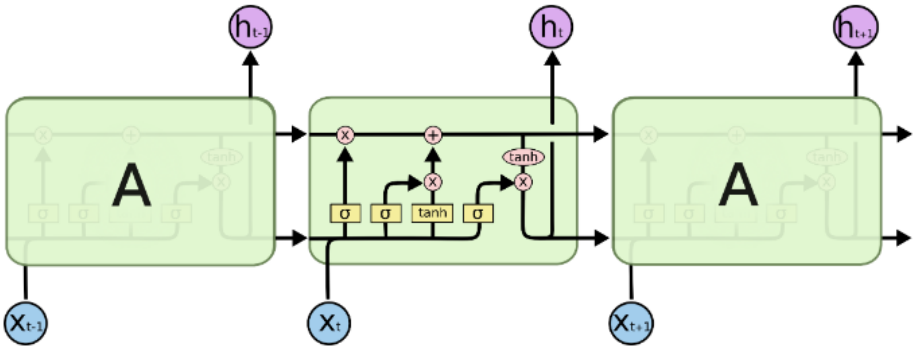

Figure 10: Model of a LSTM. Each *X* is a new input, and each *h* is an output at a time. There are four layers to determine the weight of new information versus previous information.

The extendability of LSTMs to large input sequences make it a favorable model for audio classification. Future extensions of this paper may implement RNNs and LSTMs in audio classification and compare it with the results of a standard CNN model.

Acknowledgment:

We would like to thank Dr Joe Xiao, Ph.D for suggesting the usage of the models used and explaining their compatibility with signal processing.

References:

[1]: Johnson, Tara. “What Is the Spotify Algorithm & How It Works [2022].” *Tinuiti*, 13 May 2022, https://tinuiti.com/blog/performance-display/spotify-algorithm/.

[2]: Krizhevsky, Alex. “The CIFAR-10 Dataset.” *CIFAR-10 and CIFAR-100 Datasets*, 2009, https://www.cs.toronto.edu/~kriz/cifar.html

[3]: Olteanu, Andrada. “GTZAN Dataset - Music Genre Classification.” *Kaggle*, 24 Mar. 2020, https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification.

[4]: Yang, Y.-Y., Hira, M., Ni, Z., Chourdia, A., Astafurov, A., Chen, C., Yeh, C.-F., Puhrsch, C., Pollack, D., Genzel, D., Greenberg, D., Yang, E. Z., Lian, J., Mahadeokar, J., Hwang, J., Chen, J., Goldsborough, P., Roy, P., Narenthiran, S., Watanabe, S., Chintala, S., Quenneville-Bélair, V, & Shi, Y. (2021). *TorchAudio: Building Blocks for Audio and Speech Processing*

[5]: “Sampling (Signal Processing).” *Wikipedia*, Wikimedia Foundation, 9 Oct. 2022, https://en.wikipedia.org/wiki/Sampling_(signal_processing).

[6]: “Nyquist–Shannon Sampling Theorem.” *Wikipedia*, Wikimedia Foundation, 28 Oct. 2022, https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem.

[7]: “Waveform.” *Wikipedia*, Wikimedia Foundation, 12 Aug. 2021, https://en.wikipedia.org/wiki/Waveform.

[8]: LeCroy, Teledyne. “Advantages of Two Channels in Waveform Generators.” *Mouser*, https://my.mouser.com/applications/instrumentation-two-channel/.

[9]: “Spectrogram.” *Wikipedia*, Wikimedia Foundation, 29 Aug. 2022, https://en.wikipedia.org/wiki/Spectrogram.

[10]: *Getting to Know the Mel Spectrogram - towards Data Science*. https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0.

[11]: “Gradient Descent.” *Wikipedia*, Wikimedia Foundation, 5 Oct. 2022, https://en.wikipedia.org/wiki/Gradient_descent.

[12]: “Backpropagation.” *Wikipedia*, Wikimedia Foundation, 11 Oct. 2022, https://en.wikipedia.org/wiki/Backpropagation

[13]: “Convolutional Neural Network.” *Wikipedia*, Wikimedia Foundation, 19 Oct. 2022, https://en.wikipedia.org/wiki/Convolutional_neural_network

[14]: “Recurrent Neural Network.” *Wikipedia*, Wikimedia Foundation, 30 Oct. 2022, https://en.wikipedia.org/wiki/Recurrent_neural_network

[15]: “Long Short-Term Memory.” *Wikipedia*, Wikimedia Foundation, 8 Sept. 2022, https://en.wikipedia.org/wiki/Long_short-term_memory